AN EVALUATION OF TURBO PROLOG WITH AN EMPHASIS ON ITS APPLICATION TO THE DEVELOPMENT OF EXPERT SYSTEMS

Richard B. Loftin, Ph.D.
Associate Professor of Physics
Department of Natural Sciences
University of Houston-Downtown
One Main Street
Houston, TX 77002

Turbo Prolog is a recently-available, compiled version of the programming language Prolog (Programming in Logic), originally developed at the University of Marseilles in the period from 1972 to 1974. Turbo Prolog is designed to provide not only a Prolog compiler, but also a program development environment for the IBM Personal Computer family.

An evaluation of Turbo Prolog was made, comparing its features to other versions of Prolog and to the community of languages commonly used in artificial intelligence (AI) research and development. Three programs were employed to determine the execution speed of Turbo Prolog applied to various problems: (1) a program which computes the factorial of a given integer was used to test the execution speed of Turbo Prolog with a purely computational problem, (2) the "Towers of Hanoi" was used to evaluate the speed of Turbo Prolog in executing a simple but intensely-recursive problem, and (3) the NASA benchmark planning problem (the "monkey and bananas" problem) was used to test the speed of Turbo Prolog with a problem used by NASA in its own evaluation tests[1].

The results of this evaluation demonstrated that Turbo Prolog can perform much better than many commonly-employed AI languages for numerically- intensive problems and can equal the speed of development languages such as OPS5+ and CLIPS, running on the IBM PC family of computers, with the NASA benchmark program. Applications for which Turbo Prolog is best suited include those which (1) lend themselves naturally to backward-chaining approaches (*e.g.*, theorem proving), (2) require extensive use of mathematics, (3) contain few rules, (4) seek to make use of the windowing/color graphics capabilities of the IBM PC, and/or (5) require linkage to programs in other languages (*e.g.*, C, Pascal, FORTRAN, or Assembler) to form a complete executable image.

[1]G.D. Riley, "Timing Tests of Expert System Building Tools" and "Availability of an Expert System Tool", NASA Memos FM7(86-51) and FM7(86-117).

NASA Colleague: Robert T. Savely FM72 X4751

AN EVALUATION OF TURBO PROLOG WITH AN EMPHASIS ON ITS APPLICATION TO THE DEVELOPMENT OF EXPERT SYSTEMS

Richard B. Loftin, Ph.D.
Associate Professor of Physics
Department of Natural Sciences
University of Houston-Downtown
One Main Street
Houston, TX 77002

Introduction

Two of the tasks of the Artificial Intelligence (AI) Section of the Technology Development and Applications Branch, Mission Support Directorate, Johnson Space Center, are (1) the evaluation and development of AI software for building expert systems and (2) the evaluation of AI languages. A recently-available product (May, 1986), Turbo Prolog[1] offers both a new version of an AI language and a programming environment for building expert systems. The goals of the project described in this report were (1) the evaluation of Turbo Prolog as an AI language and (2) the production of benchmark programs, written in Turbo Prolog, which permit Turbo Prolog's execution speed to be directly compared with that of alternatives already evaluated by the AI Section[2].

In order to achieve the first goal, time was devoted to a study of Turbo Prolog in the context of other versions of Prolog and the development of simple programs using this language. Two simple tests of Turbo Prolog's execution speed were made using the computation of factorials and the Towers of Hanoi. The final benchmark program was of the standard type used by the AI Section in evaluating the speed of a number of expert system development tools[3]. The problem is one of proceeding to a prescribed goal by means of subgoals which must be achieved first. Initial conditions are supplied and approximately thirty rules specify the manner in which the subgoals and the final goal may be satisfied. This benchmark has been written and implemented in a variety

of languages on a variety of computers. By comparing the speed with which this benchmark program executes when written in Turbo Prolog with the same benchmark in different programming languages running on the same computer, a measure of Turbo Prolog's efficacy as a language for the development of expert systems can be had.

This report begins by discussing the history of Prolog and continues by presenting the major features of Turbo Prolog, emphasizing those which set it apart from other versions of the language. Finally, the benchmark timing results are presented and come conclusions are drawn regarding the use of Turbo Prolog as a tool in the development of expert systems.

A Brief History of Prolog

The origins of Prolog (Programming in Logic) can be traced back to the 1965 publication of the Resolution Principle by J. A. Robinson[4]. During the early 1970's a number of workers attempted to implement languages that embodied logic[5,6,7,8,9,10]. Kowalski's development of predicate calculus in 1972[11] added a powerful tool to the kits of those seeking to produce languages that were oriented toward theorem proving. It was the collaborative efforts of R. A. Kowalski and Alain Colmerauer during the year Kowalski spent at the University of Marseilles that led to the development of Prolog's specifications[12] in 1972. Colmerauer and his coworkers at Marseilles quickly began to implement these specifications and produced the first interpreters in 1973[13,14]. With the detailed publication of Prolog's specifications in and of its implementation in 1975[15], other university groups began to use the "Marseilles" Prolog and began to develop their own Prolog versions[16,17,18,19,20]. It was the publication of Programming in Prolog by Clocksin and Mellish in 1981[21] that brought some order to the proliferation of dialects of Prolog. By 1984, with the appearance of the second edition of Clocksin and Mellish[22], most users of Prolog were accustomed to a common syntax and grammar for the language.

The announcement by Japan in 1982[23] that Prolog would be the

language for their "fifth-generation" project catapulted Prolog, until that time a predominantly European institution, into international prominence. Until recently most U.S. AI practitioners have eschewed the use of Prolog in favor of Lisp, in large measure due the the availability of powerful development environments for Lisp machines. The advent of Turbo Prolog may well serve to introduce Prolog into the "mainstream" of computing in the U.S. It provides a powerful and inexpensive (<$100) development environment for Prolog utilizing an extremely popular personal computer family--the IBM PC/XT/AT).

Features of Turbo Prolog

Naturally, the feature that sets Turbo Prolog (and, for that matter) all Prologs apart from other AI languages is its backward-chaining nature. Most commonly used expert system development tools are implemented with forward-chaining, although some, like KEE and ART, can employ backward-chaingin also. At first glance Turbo Prolog seems to have embraced the syntax and functionality of the "standard" set by Clocksin and Mellish[22]. Syntactically, this is more "almost" correct. Important differences exist, however, which are pitfalls for the experienced Prolog programmer. One essential difference (from which flows many "subdifferences") is the typed nature of the Turbo Prolog compiler. In this instance Turbo Prolog resembles FORTRAN or Pascal--each domain's type must be declared, either in the "domain" section or in the declaration of a predicate. This single feature sets Turbo Prolog apart from other versions of Prolog and from most other AI languages in general. It is both a weakness and a strength. There is no doubt that much of the speed and error checking power of the compiler is due to domain typing. Experienced AI programmers are not accustomed to a requirement that domains be typed. It is common to have functors, for example, whose arguments may change from integer to real as a result of a clause. In Turbo Prolog this means that each possible argument type must be declared at the time the program is written. Additional "deltas" with other Prolog versions also exist. For example, "=" is not the unification operator of Clocksin and Mellish, rather it is more like the "is" operator; commas do not act as operators; the programmer cannot define his own infix operators; the result of an arithmetic operation depends on

the type(s) of the arguments; operators cannot be passed as functors; and missing are the standard predicates arg, functor, clause, univ, and op. Turbo Prolog unfortunately lacks a virtual database support and database predicates are not executable.

Figure 1 shows the structure of a Turbo Prolog program. The elements that are enclosed in brackets are optional. The program section is used if this program is to linked to others (written in Prolog, C, FORTRAN, Pascal, or Assembler) to form an executable whole. The directives section is used to issue orders to the compiler (for example, invoking the trace facility or declaring the amount of memory to be allocated to the code). The domains section is used to declare the types of all predicate arguments (it may be omitted if there are no compound predicates and the type declaration can be included in the predicate section). Global domains are used for those predicates that will be

PROLOG PROGRAM STRUCTURE

[PROGRAM]

[DIRECTIVES]

DOMAINS

[GLOBAL DOMAINS]

[DATABASE]

PREDICATES

[GLOBAL PREDICATES]

CLAUSES

[GOAL]

FIGURE 1: The Structure of a Turbo Prolog Program

accessed by other programs linked to the present one and the database section is used to identify those predicates that will be changed by "assert" during program execution. The predicate section contains a list of all predicates and their arguments and the global predicate section serves the same function as the global domains section. Clauses are listed in the clause section. Goals may be declared in the program itself. If the goal section is missing, Turbo Prolog prompts the user for a goal in the dialogue window.

In "giving" up some of the familiar features of other Prologs, the user of Turbo Prolog does gain a great deal. Unlike most AI languages, Turbo Prolog contains a complete set of arithmetic and trigonometric operators. In addition, there are about thirty "new" standard predicates that allow the programmer to access the full range of power of the IBM PC family. For example, Turbo Prolog contains a complete set of graphics commands for the PC, including windowing and the ability to mix text and graphics in the same window. Sound and color are both supported as well as input/output via files, devices, or ports. Turbo Prolog allows the programmer to link a prolog program to other programs written in C, FORTRAN, Pascal, or Assembler. The programmer (as well as the user of a developed application) has full access to DOS, BIOS, and the built-in Turbo editor. Perhaps the "nicest" thing provided by Turbo Prolog is a powerful development environment, based on the PC, that is extraordinarily inexpensive compared with those used by most AI programmers. The development environment provides four windows (the user controls the size and foreground/background color of each window): editor, dialogue, trace, and message. A banner menu is provided allowing the user to select editor, run (compiles and runs), compile (allows the user to compile to an object or executable file), options (selects whether the compilation is to an object or executable file), setup (allows the user to configure the windows, define directories, and perform other useful "housekeeping" tasks) and quit (which returns the user to DOS). The editor is a "full-window" editor and uses the commands of Wordstar. The compiler, like that of Turbo Pascal, stops when an error in encountered, returns to the programmer to the editor, and places the cursor at the location of the error. The powerful trace facility allows the user to examine every call and return for the entire program or for selected

clauses. All-in-all, Turbo Prolog is a pleasant way to quickly develop executable code.

Benchmarks

Three benchmarks were chosen to measure the speed of execution of a Turbo Prolog program in performing three very different tasks. To test Turbo Prolog's execution speed with arithmetic operations, a simple program was used to compute the factorial of an integer (Appendix A contains the source code for this program). The program was run on an IBM PC and an IBM PC/AT to compute the factorial of 170 (the result of this computation is near the capacity of the PC). The time required for this computation is shown in Table 1. The Towers of Hanoi problem provides another benchmark program which is intensively recursive and makes large demands on the stack (Appendix B contains the source code for this program). The times required for the execution of this program with different numbers of disks are also included in Table 1.

The final benchmark was chosen to permit the speed of Turbo Prolog to be directly compared to that of other expert system tools in the execution of a rule-based expert system. The problem tackled was a variation of the well-known "monkey and bananas" problem[24]. This particular variation was developed by the AI Section as a means of comparing a large number of expert system development tools[2,3]. The general problem is prototypical of a number of planning problems in which many subgoals must be identified and reached in order for the "main" goal to be achieved.

The monkey and bananas program, as implemented in Turbo Prolog (the source code for the program is contained in Appendix C) consists of 34 "rules" in the form of clauses or subclauses. A total of 22 predicates were used. Table 2 contains the time required for execution of this program on both the IBM PC and IBM PC/AT. The table also contains the accumulated timing tests obtained by the AI Section through the end of July, 1986. It should be noted that there is some ambiguity in determining the execution speed of a Turbo Prolog program. After compilation is complete, but before execution begins, Turbo Prolog

TABLE 1: Execution Times for Two Benchmarks Using Turbo Prolog (See Appendices for Source Code)

Benchmark	Execution Times (s)	
	IBM PC	IBM PC/AT
Factorial of 170	0.27	0.10
Towers of Hanoi		
3 Disks	<0.005	----
10 Disks	0.27	0.10
12 Disks	1.20	0.43
15 Disks	9.55	3.40
16 Disks	----	6.86

checks the program's clauses against the given goal(s). Those clauses which will be called in order to reach the given goal(s) are selected through this "preprocessing". Only after this is accomplished is the goal actually executed. This means that the internal time function can only be accessed after the preprocessing is complete. Since the user is normally concerned with the total "run" time, which includes both preprocessing and execution, it is this run time which is reported. A footnote gives the measured execution times for the program running on both machines tested.

Conclusions

Turbo Prolog may be, in the view of at least one evaluator[25], not so much another version of Prolog, as a new language in itself. Turbo Prolog has proven to be exceptionally easy to begin to use and Borland has "encased" it in an superb development environment. The syntax of the language and its standard predicates depart significantly from the Prolog "standard"; this may pose a barrier to the experienced Prolog programmer,

Table 2: Timing Tests of Expert System Tools for the NASA "Monkey and Bananas" Benchmark*

TOOL(VERSION)	MACHINE	TIME(S)
ART(V2.0)	SYMBOLICS	1.2
ART(V2.0)	TI EXPLORER	2.4
ART(V2.0 BETA)	LMI	3.0
ART**	SYMBOLICS	7.6
ART(V BETA 3)	VAX	17
CLIPS(V3.0)	SUN	1.2
CLIPS(V3.0)	VAX	2.5
CLIPS(V3.0)	HP9000	4.0
CLIPS(V3.0)	IBM PC/AT	7.0
CLIPS(V3.0)	IBM PC	21.1
ExperOPS5(V1.04)	MACINTOSH	55
KEE(V2.1.66)**	SYMBOLICS	17.8
KEE(V2.2.66)	SYMBOLICS	165
OPS5(VAX V2.0)	VAX	1.3
OPS5(FORGYVPS2)	SYMBOLICS	1.7
OPS5+(V2.0003)	IBM PC/AT	5.2
OPS5+(V2.0002)	MACINTOSH	14
OPS5+(V2.0002)	IBM PC	19
OPS83	VAX	0.46
OPS83	IBM PC/AT	1.1
OPS83	IBM PC	3.3
TURBO PROLOG	IBM PC/AT	6.73***
TURBO PROLOG	IBM PC	20.43***

*SOURCES (EXCEPT TURBO PROLOG): NASA MEMOS FM7(86-51) AND FM7(86-117)

**IMPLEMENTED USING BACKWARD-CHAINING RULES

***"RUN TIME", EXECUTION TIMES are 0.21S AND 0.65 FOR PC/AT AND PC

but others (especially fans of Turbo Pascal) will appreciate the "unique" features of Turbo Prolog. Those predicates which are missing from Turbo Prolog are either seldom used or their function can be achieved in other ways. The superb programming environment (convenient editor, powerful trace facility, compiler, built-in mathematical functions, and access to IBM PC features such as graphics, windows, color, sound, and I/O through ports or files) coupled with its inexpensive cost makes Turbo Prolog an attractive tool for those who have not tackled an AI language before. For those developing expert systems, Turbo Prolog may prove to be well-suited for fast prototyping of "small" rule bases or for those applications that lend themselves to backward-chaining approaches (for example, theorem proving). Surprisingly, Turbo Prolog executes the NASA benchmark as fast as popular expert system development tools like OPS5+ or NASA's own CLIPS.

Appendices

The appendices mentioned in the body of this paper are not included with the published report due to their length. Copies of these appendices may be obtained directly from the author or from his NASA/JSC colleague, Robert T. Savely (NASA/Johnson Space Center, Mail Code FM72, Houston, TX 77058).

REFERENCES

1. Turbo Prolog is a product of Borland International, Inc., 4585 Scotts Valley Drive, Scotts Valley, CA 95066.

2. Gary D. Riley, "Timing Tests of Expert System Building Tools," NASA/ Johnson Space Center, Memo FM7(86-51); Robert T. Savely, "Availability of an Expert System Tool," NASA/Johnson Space Center, Memo FM7 (86-117).

3. Gary D. Riley, "Benchmarking Expert System Tools," in Proceedings of Robex '86 (The Second Annual Workshop on Robotics and Expert Systems), held at NASA/Johnson Space Center, 4-6 June 1986, p. 61.

4. J. A. Robinson, "A Machine-Oriented Logic Based on the Resolution Principle," J. ACM 12(1), 23(1965).

5. G. Sussman and D. V. McDermott, "MICRO-PLANNER Reference Manual," AI Memo 203, AI Laboratory, MIT, 1970.

6. R. M. Burstall, J. S. Collins, and R. J. Popplestone, Programming in POP-2, Edinburgh: Edinburgh University Press, 1971.

7. C. Hewitt, "Description and Theoretical Analysis (using schemata) of PLANNER, a Language for Proving Theorems and Manipulating Models in a Robot," Report No. TR-258, AI Laboratory, MIT, 1972.

8. D. J. M. Davies, "POPLER: A POP-2 Planner," Rep. No. MIP-89, School of AI, University of Edinburgh, 1972.

9. G. Sussman and D. V. McDermott, "CONNIVER Reference Manual," Memo 259, AI Laboratory, MIT, 1972.

10. D. Davies, et. al., POPLER 1.5 Reference Manual, Edinburgh: University of Edinburgh, 1973.

11. R. A. Kowalski, "The Predicate Calculus as a Programming Language," in Proceedings of the International Symposium and Summer School on Mathematical Foundations of Computer Science, held at Jabłonna, Poland, 1972.

12. A. Colmerauer, H. Kanoui, P. Roussel, and R. Pasero, "Un Système de Communication Homme-Machine en Français", Rapport preliminaire, Groupe d'Intelligence Artificielle, Université d'Aix-Marseille, 1972.

13. G. Battani and H. Meloni, "Interpréteur du langage de programmation PROLOG," Group d'Intelligence Artificielle, Université d'Aix-Marseille, 1973.

14. A. Colmerauer, H. Kanoui, P. Roussel, and R. Pasero, "Un système de Communication Homme-Machine en Français," Rapport de Rechereche sur le contrat CRI no 72-18 de février 72 a juin 73, Groue d'Intelligence Artificielle, Université d'Aix-Marseille, 1973.

15. P. Roussel, PROLOG, Manuel de Référence et d'Utilisation, Université d'Aix-Marseille: Groupe d'Intelligence Artificielle, 1975.

16. D. H. D. Warren, "Implementing Prolog--Compiling Predicate Logic Programs," DAI Report Nos. 39 and 40, Edinburgh, 1977.

17. D. H. D. Warren, L. M. Pereira, and F. C. N. Pereira, "Prolog--the Language and Its Implementation Compared with Lisp," presented at the ACM Symposium on Artificial Intelligence and Programming Languages, Rochester, New York, SIGART Newsletter No. 64, SIGPLAN Notices 12(8), 109 (1977).

18. J. Bendl, P. Köves, and P. Szeredi, "The MPROLOG System," in Tärnlund 1980, 201 (1980).

19. F. G. McCabe, Micro PROLOG Programmer's Reference Manual, London: Logic Programming Associates Ltd., 1981.

20. F. Kluźniak and S. Szpakowicz, Prolog, Warsaw: Wydawnictwa Naukowo-Techniczne, 1983.

21. W. F. Clocksin and C. S. Mellish, Programming in Prolog, Berlin: Springer-Verlag, 1981.

22. W. F. Clocksin and C. S. Mellish, Programming in Prolog, Second Edition, Berlin: Springer-Verlag, 1984.

23. "Outline of Research and Development Plans for Fifth Generation Computer Systems," Institute for New Generation Computer Technology (ICOT), Tokyo, May, 1982.

24. L. Brownston, R. Farrell, E. Kant, and N. Martin, Programming Expert Systems in OPS5, Reading, MA: Addison-Wesley Publishing Co., 1985.

25. D. Rubin, "Turbo PROLOG: A PROLOG Compiler for the PC Programmer," AI Expert, Premier Issue, 87 (1986).